修 士 論 文 の 和 文 要 旨

研究科・専攻	電気通信大学院　情報理工学研究科 情報・ネットワーク工学専攻　博士前期課程		
氏　　名	尾作　洋彦	学籍番号	1931034
論 文 題 目	FPGA を用いたデータベースクエリ処理に関する研究		

要　旨

多様なネットワークサービスの普及やセンサ技術の発達などに伴い，データセンターに収集され，蓄積されるデータ量は年々急速な増大を続けている．

そのような莫大なデータを解析し，重要データの抽出，意思決定，危機管理，サービスの改善などを行うビッグデータ解析が広く行われるようになった．

ビッグデータ解析の実行にあたって，蓄積されるデータの増加速度とアプリケーションの拡大は計算機の性能向上速度を超えているため，現実的な計算時間で実行するために計算機システムの規模は拡大を続けている．

計算機システムの拡大による消費電力増大への対応策の 1 つとして，GPU などのメニーコアプロセッサによる高効率な計算機システムが利用されている．

その一方で，ビッグデータ解析は演算あたりのデータ量が多く，主記憶へのデータ転送がボトルネックになりやすい．

そこで，ボトルネックを解消するためにネットワークやストレージに接続した Field Programmable Gate Array (FPGA) と呼ばれる構成変更が可能な集積回路を導入して，FPGA 上の専用ハードウェアがデータの転送経路上で演算を実行する仕組みが開発されている．

我々の研究グループでは，ビックデータ解析のためにストレージやネットワークからホストの主記憶などへデータを転送する経路上で演算を行う In-datapath Computing を開発している．In-datapath Computing を用いて，フラッシュストレージからのデータ転送経路上で，クエリに応じた集約演算を実行する専用ハードウェアを試作した．

本研究報告では，OLAP (On-Line Analytical Processing：オンライン分析処理)の有名なベンチマークの 1 つ TPC-H に対して，Impala を用いた In-datapath Computing を用いない分散処理ソフトウェア実行と実行クエリを変更可能な専用ハードウェアを実装した FPGA を比較し，性能を評価した．

TPC-H のクエリ 3 を対象とした実験では，ソフトウェア実行に比べ約 10 倍の実行速度が得られた．

令和２年度修士論文

FPGAを用いた
データベースクエリ処理に関する研究

情報理工学研究科
情報・ネットワーク工学専攻

学 籍 番 号 :　1931034
氏 名 :　尾作　洋彦
主任指導教員 :吉永　努 教授
副指導教員 :　策力木格 准教授
提出年月日 :　令和3年1月25日

(表紙裏)

目 次

図 目 次

表 目 次

第1章　序論

多様なネットワークサービスの普及やセンサ技術の発達などに伴い，データセンターに収集され，蓄積されるデータ量は年々急速な増大を続けている．そのような莫大なデータを解析し，重要データの抽出，意思決定，危機管理，サービスの改善などを行うビッグデータ解析が広く行われるようになった [1]．ビッグデータ解析の実行にあたって，蓄積されるデータの増加速度とアプリケーションの拡大は計算機の性能向上速度を超えているため，現実的な計算時間で実行するために計算機システムの規模は拡大を続けている．計算機システムの拡大による消費電力増大への対応策の 1 つとして，GPU などのメニーコアプロセッサによる高効率な計算機システムが利用されている．その一方で，ビッグデータ解析は演算あたりのデータ量が多く，主記憶へのデータ転送がボトルネックになりやすい．そこで，ボトルネックを解消するためにネットワークやストレージに接続した Field Programmable Gate Array (FPGA) と呼ばれる構成変更が可能な集積回路を導入して，FPGA 上の専用ハードウェアがデータの転送経路上で演算を実行する仕組みが開発されている．我々の研究グループでは，ネットワークとストレージを密に接続する FPGA ボードを搭載した PC を，複数ネットワーク結合して構築した PC-Cluster をビッグデータ解析の高速計算基盤とする Interconnected-FPGAs と呼ぶシステムの研究開発に取り組んでいる [2]．Interconnected-FPGAs を用いたビッグデータ解析高速化の応用例として，データベースの集約演算を実行するハードウェアを実装した [3]．これらの高速化の仕組みにより，OLAP (On-Line Analytical Processing : オンライン分析処理) の有名なベンチマークの 1 つ TPC-H[4] のクエリ 1 を対象とした性能測定の結果，データベースソフトウェアに対する性能向上が確認されている [2]．また，マルチ FPGA 環境においても良いスケーラビリティが得られることを確認している．本研究報告では，実行クエリを変更可能な専用ハードウェアを FPGA に実装し，Impala を用いたソフトウェア実行と比較し，性能を評価した．TPC-H のクエリ 3 を対象とした実験では，ソフトウェア実行に比べ約 10 倍の実行速度が得られた．

第2章　関連研究

2.1　ビッグデータ解析と専用ハードウェアによる高速化

2.1.1　ビックデータとビッグデータ解析

ビッグデータとは，単に量が多いだけでなく，様々な種類・形式が含まれる非構造化データ・非定型的データのことを指す．ビッグデータ解析では，そのような莫大なデータを解析し，重要データの抽出，意思決定，危機管理，サービスの改善などを行う．図 2.1.2 に示すようにビッグデータのデータ量は年々増大しており，現実的な計算時間で解析を行うために処理をできるだけ高速に行うことが求められている．

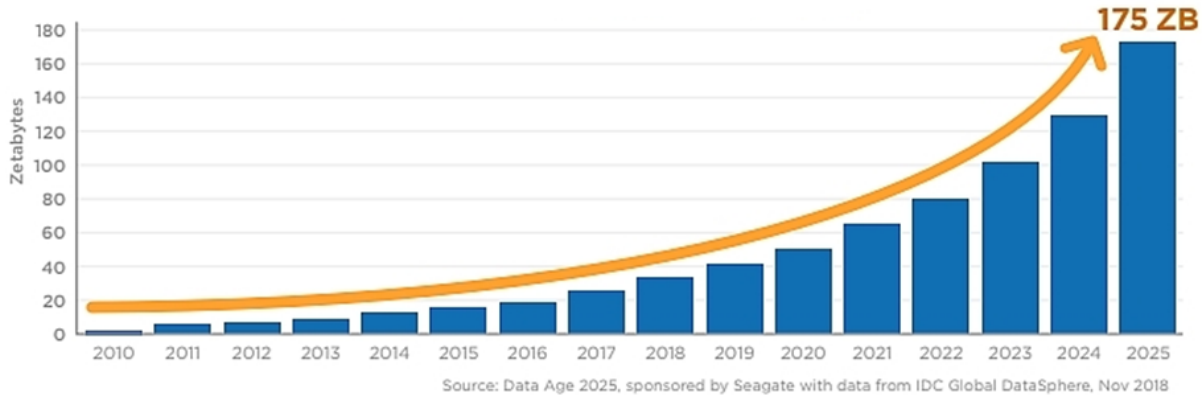

図 2.1.1: Annual Size of the Global Datasphere

2.1.2　専用ハードウェアによる高速化

現実的な計算時間で解析を行う方法の 1 つとして，ネットワークやフラッシュストレージからのデータの転送経路上で FPGA などの計算素子を配置し，高速化を図る研究が存在する [3]．この研究では，図**??**に示すようにストレージから読み出したデータに対して，集合演算などのクエリ処理を専用ハードウェアで肩代わりすることで，ホストの負荷の軽減及び転送データ量の削減を行うことで高速化を実現している．また，ビッグデータ解析に FPGA による専用ハードウェアを導入して高速化を図る研究には，複数の研究機関が取り組んでいる．IBEX と呼ぶ基本的な集約演算機能を持つクエリ処理のオフロードをサポートするストレージエンジンが存在する [5]．ただし，実行可能なクエリの柔軟性はあまり高くない．IBM 社による Netezza と呼ばれる，FPGA をストレージとプロセッサの間に組み込んだブレードサーバが存在する [6]．Netezza は，ストレージからのデータ読み出し時に，FPGA で不必要なデータを削除し，必要なデータだけを抽出すること

で，高速なデータベース処理を実現している．Lai らは，我々が扱うビッグデータ解析とは異なるビッグデータ解析のソーティングに対して，水平方向と垂直方向の両方でソートする 2 方向ソートアーキテクチャを提案し，FPGA に実装することでコストのかかるデータ送信時間の大幅な削減を実現した [7]．また，Li らはストレージノードのデータクエリ効率を加速し，コンピューティングノードのワークロードとそれらの間の通信オーバーヘッドを削減するために，クエリフィルタ (NIC-QF) を備えた FPGA ベースのネットワークインターフェイスカードを提案した [8]．データをストレージノードに格納し，コンピューティングノードへ送信するというアプローチが我々の研究とは異なっている．飯塚らは，FiC (Flow in Cloud) と呼ぶ複数台の FPGA を高速シリアルリンクで接続した FPGA クラスタを構築し，クラウドの計算基盤とする研究を勧めている [9]．ここでは，畳み込みニューラルネットワーク (CNN) の AlexNet を実装し，評価している．その他にも，FPGA のアーキテクチャのみで高速化するのではなく，CPU と FPGA がハイブリッドに合わさったアーキテクチャ上でデータベース演算を高速化するフレームワークである Centaur[10] が挙げられる．

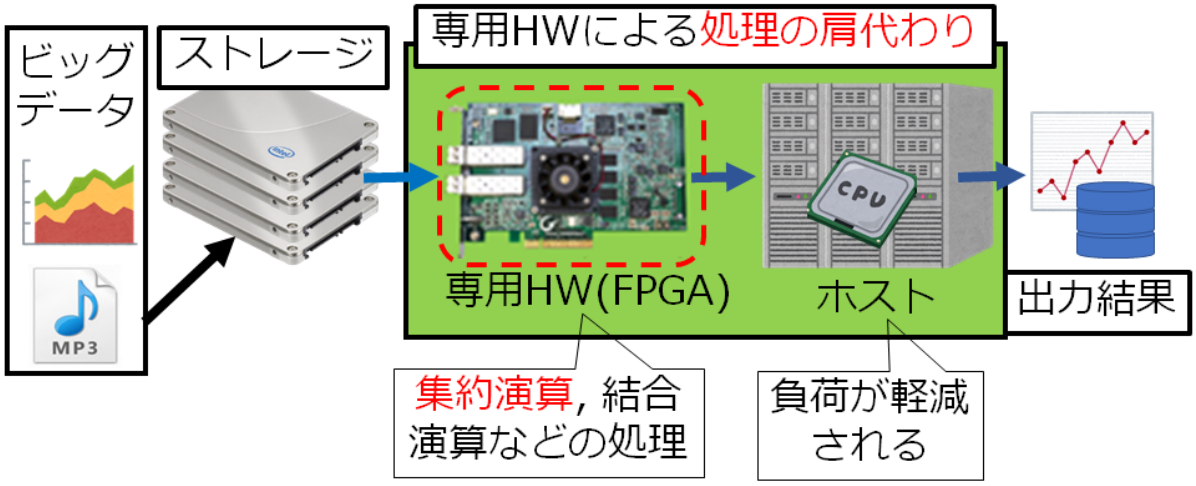

図 2.1.2: 専用ハードウェアを用いたビッグデータ処理

第3章　設計方針

本研究報告では，FPGA で構成した専用ハードウェアでクエリ処理の一部を行い，主記憶へ転送するデータ量を削減し，集約演算を実行するシステムを提案する．また，実行するクエリの柔軟性を高めるために，実行するクエリを変更可能な機能を実装する．このシステムを高速化するための仕組みおよびシステムが対象とするクエリ，そのクエリの実行方式，専用ハードウェアの設計方針を以下に示す．

3.1　In-datapath Computing

FPGA 内部に高速化を実現する仕組みとして，In-datapath Computing と呼ぶ機能を搭載する．これは，ストレージやネットワークからホストの主記憶などへのデータ転送経路上で演算を行う機能である．図 3.1.1 に FPGA を組み込んだ計算ノードの構成を示す．主記憶にデータを転送してプロセッサによって処理される部分を FPGA 上にオフロードすることで，プロセッサの負荷を下げることが可能である．

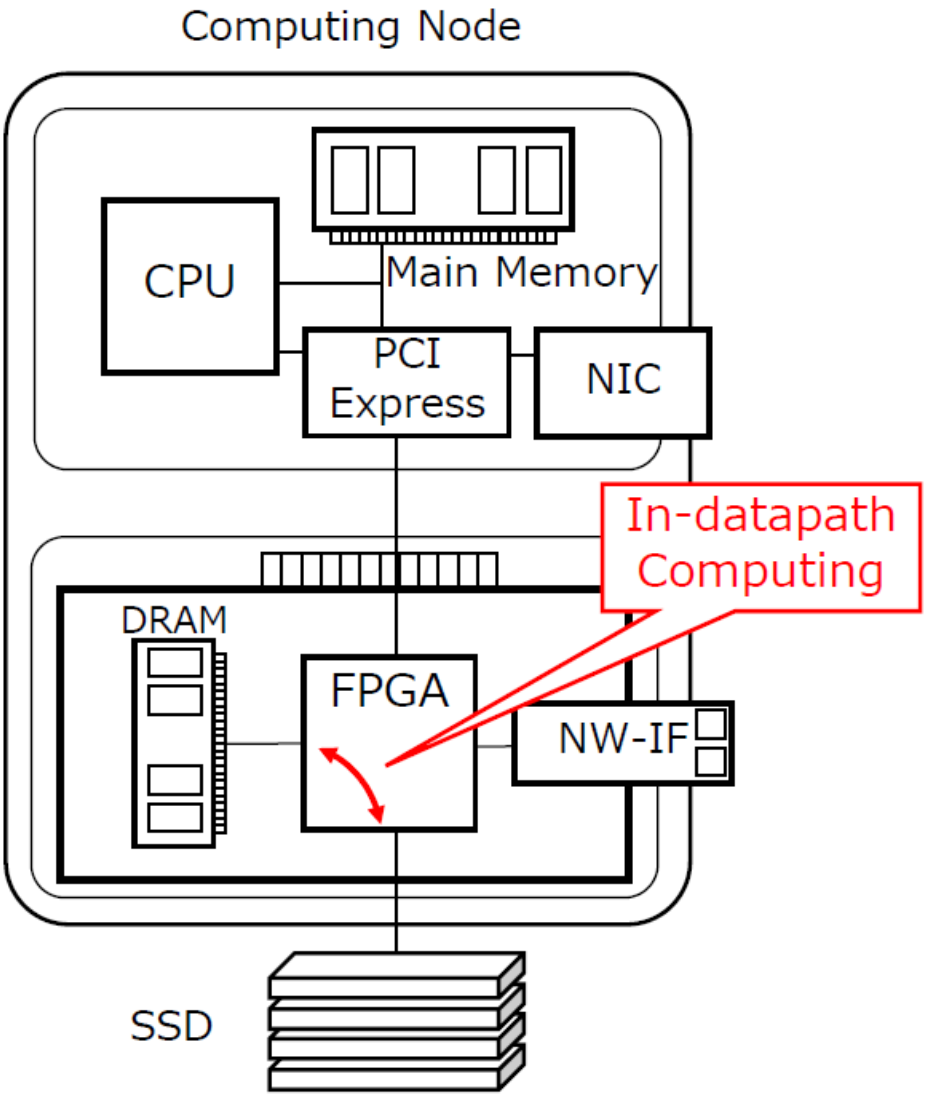

図 3.1.1: 計算ノードの構成

3.2 対象クエリ

集約演算を含むクエリを対象とする．それに伴い，データのフィルタリングおよび論理演算，算術演算といったクエリ処理も行う．SQL 文における GROUP BY 句および SELECT 句,WHERE 句, ORDER BY 句のみを含むクエリである．

3.3 クエリの実行方式

実行するクエリの作成および設定フローを図 3.3.1 に示す．SQL 文で記述した対象クエリをクエリコンパイラに入力し，FPGA 上に実装する専用ハードウェアで実行すための命令列を生成する．FPGA 上のクエリ処理は，生成した命令列を FPGA 内部のローカルレジスタに格納し，5.3 節で説明する PRA モジュールがこの命令列を実行することで実現する．ローカルレジスタに格納する命令列を変更することで FPGA で実行するクエリを変更することができる．また，FPGA に SATA インタフェースを搭載し，クエリの対象データを二次記憶（SSD) から直接読み込みできるようにする．FPGA に搭載する SATA インタフェースの速度でデータを読みながら，実行するクエリの対象データのフィルタリングや集約演算を実行する．これらは，FPGA の構成を再構成することなく実行クエリを変更可能にするための方式である．

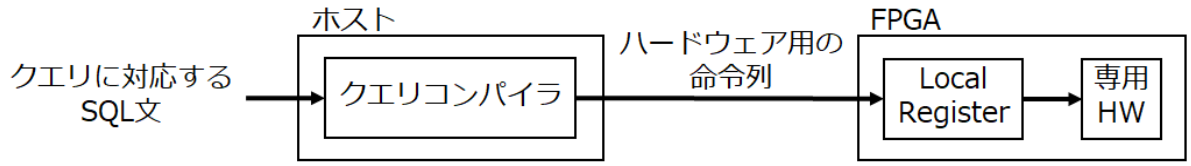

図 3.3.1: クエリの作成および設定フロー

3.4 専用ハードウェアの設計方針

3.2 節で述べた対象クエリを図 3.4.1 に示す 3 段のパイプラインで処理する．パイプライン処理を行わない場合，SSD からのデータ供給スピードに対して，クエリ処理スピードが律速となってしまう．そのため，SSD からのデータ読み出しのワイヤースピードでクエリ実行を行えるように，このように設計する．パイプラインの各ステージは，それぞれカラム抽出，タプル抽出，集約演算を実行する．カラム抽出ステージでは，SSD から読み出した表データから対象のクエリ処理に必要なカラムデータに対してデータのフィルタリングを行うことによって，クエリ処理に不要な表データは切り捨て，必要なカラムデータの抽出を行う．次に，タプル抽出ステージでは，カラム抽出したデータに対して対象クエリに指定された算術論理演算を適用することで，データのフィルタリングを行い，集約演算に必要なタプルデータを抽出する．最後に，集約演算ステージでは，抽出されたタプルデータに対して GROUP BY 集約演算を実行する．

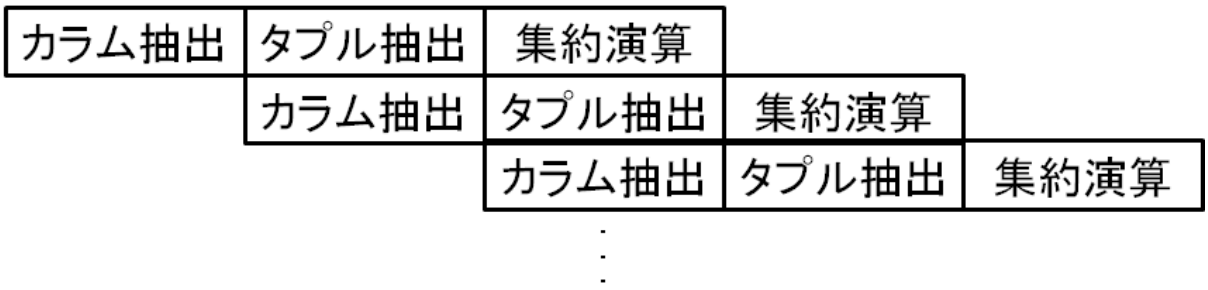

図 3.4.1: 集約演算クエリ処理パイプライン

第4章　ストリームデータに対するクエリ処理

3.4 節で述べた設計を実現するパイプラインの各ステージのクエリ処理内容について，図 4.3.1 の右図に示す TPC-H クエリ 1 を改良したクエリ 1 改の SQL 文を例として以下に詳しく説明する．除算器のリソース量と計算にかかる時間を考慮し，集約演算における AVG (平均値) を本研究では対象外としているため，クエリ 1 改は，ソフトウェア側で行う必要がある AVG の演算を省いたクエリとなっている．また，図 4.3.1 の左図に示す表データを図 4.0.1 に示した 16 バイトで構成するチャンクと呼ぶデータ列へと変換し，複数束ねた入力タプルとする．さらに右図に示すように複数の入力タプルで入力データを構成する．各ステージにおける処理はタプル単位で実行する．

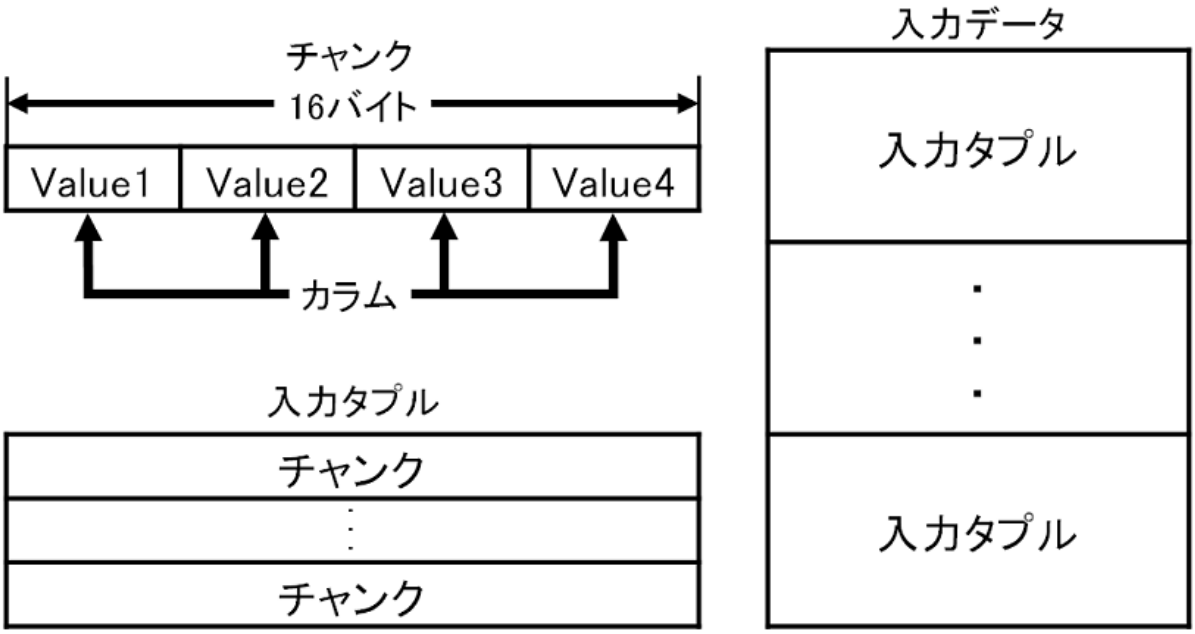

図 4.0.1: 入力データ構成

4.1 カラム抽出ステージ

カラム抽出ステージでは，入力データから実行するクエリの SQL 文の SELECT 句および WHERE 句，GROUP BY 句に書かれた必要なデータのフィルタリングを行うことでカラムの抽出を実行する．カラムの抽出はチャンクごとに順に確認する．図 4.3.1 の左図に示す表データの赤枠で囲まれた部分および図 4.3.1 の右図に示す SQL 文の赤マーカーで塗られた部分が必要なカラムである．このクエリの場合は，l_returnflag および l_linestatus, l_quantity, l_extendedprice, l_discount, l_tax, l_shipdate の 7 つである．必要なカラムがないチャンクは，タプル抽出ステージへ出力せず破棄し，あるチャンクはチャンク内で不要なカラムがあれば 0 データで上書きした上で，タプル抽出ステージへ出力する．最終チャンクを確認後，次の入力タプルの確認に移る．

4.2 タプル抽出ステージ

タプル抽出ステージでは，カラム抽出したデータに対して，対象クエリの算術論理演算に基づいてデータのフィルタリングを行い，タプルの抽出を実行する．まず，論理演算に対する条件判定を行うが，乗算があるクエリの場合は乗算に時間がかかるため，乗算 1 つを先に開始する．図 4.3.1 の右図の SQL 文の例ではまず，論理演算による条件判定を行う前に，緑枠で囲まれた SELECT 句に書かれた乗算の 1 つを開始する．続いて，論理演算による条件判定を行う．この例では，図 4.3.1 の右図の緑枠で囲まれた WHERE 句に書かれた条件に対して真偽を判定する．偽であった場合は現在のタプルを破棄し，次のタプルの条件判定へと移る．真であった場合は，タプルを破棄せず集約演算に必要なデータの集約演算ステージへの出力および行っていない算術演算の実行へと移る．集約演算に必要なデータの内，算術演算が必要のないデータは集約演算 KEY とともに集約演算ステージへと適宜出力する．図 4.3.1 の右図の橙枠で囲まれた GROUP BY 句に書かれた l_returnflag および l_linestatus が集約演算 KEY であり，SELECT に書かれた l_quantity と l_extendedprice が算術演算が必要のないデータに該当する．続けて，まだ行っていない算術演算を行い，SQL 文 1 行にあたる算術演算終了ごとにその結果と集約演算 KEY を合わせて集約演算ステージへと出力する．全ての算術演算終了後，次のタプルの処理へと移る．

4.3 集約演算処理

集約演算処理では抽出したタプルに対して集約演算 KEY に従い集約演算を実行する．集約演算は SUM (合計値) および MAX (最大値)，MIN (最小値)，AVG (平均値)，COUNT (出現数) を求める演算である．この例では，図 4.3.1 の右図の青枠で囲まれた SELECT 句に書かれた SUM および COUNT の集約演算を行う．ただし，AVG の演算は除算器のリソース量と計算にかかる時間を考慮し，本研究では FPGA での演算を行わず，ソフトウェア側で SUM / COUNT の除算によって計算する．

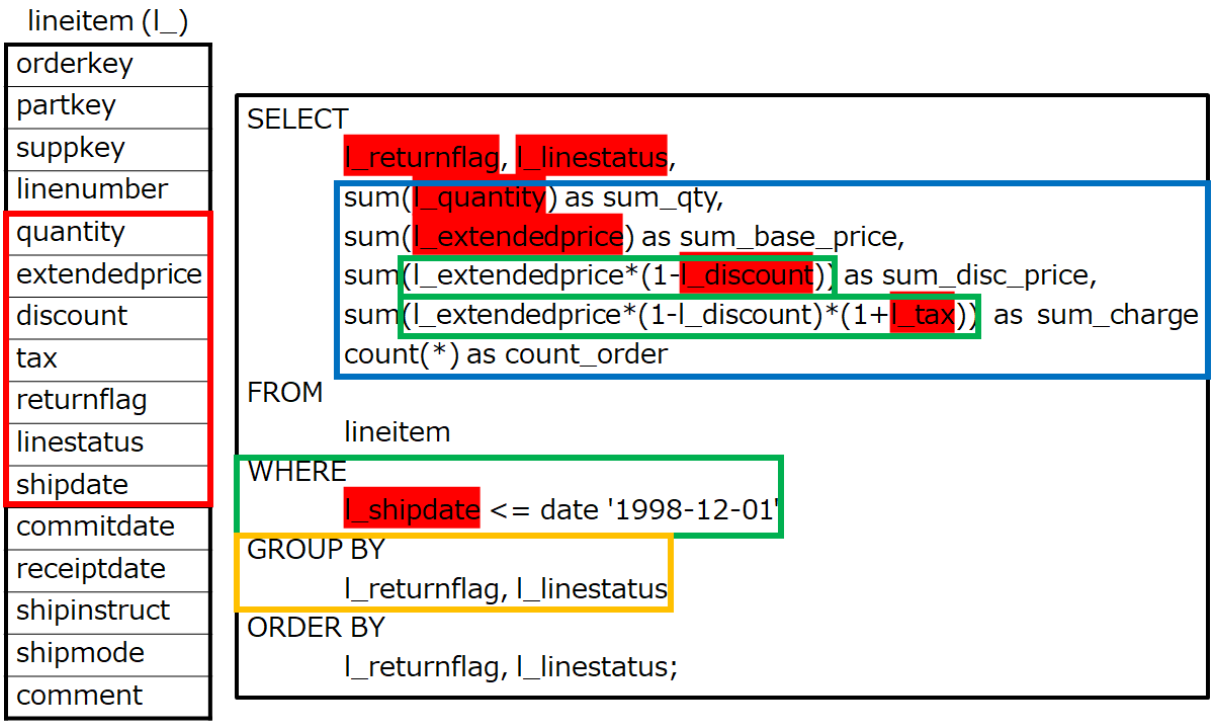

図 4.3.1: lineitem テーブル (左図) およびクエリ 1 改の SQL 文 (右図)

第5章　ハードウェア実装

4節で述べたクエリ処理を実現するハードウェアモジュールを以下に示すFPGAボード上に実装する．また，FPGA内に高速化を実現する仕組みを搭載する．

5.1　FPGAボードAPX-7142改

アバールデータ社のAPX-7142[11]を改造したAPX-7142改と呼ぶFPGAボードを使用する．FPGAボードの外観を図5.1.1に，FPGAボードの仕様を表5.1に示す．APX-7142改は，APX-7142にSASポートを追加し，4台のSSDを接続できるようにしたものである．ストライピングによって4台のSSDに同時アクセスでき，データの読み書きを高速に行うことが可能である．各FPGAボードはPCI-Expressでホストと接続され，ホスト上のAPIを介してFPGAボード上のDRAMやFPGA内のローカルレジスタにアクセス可能である．また，FPGA内の各モジュールは，AVAL-BUSと呼ばれるアバールデータ社独自のバス規格を用いて通信する．

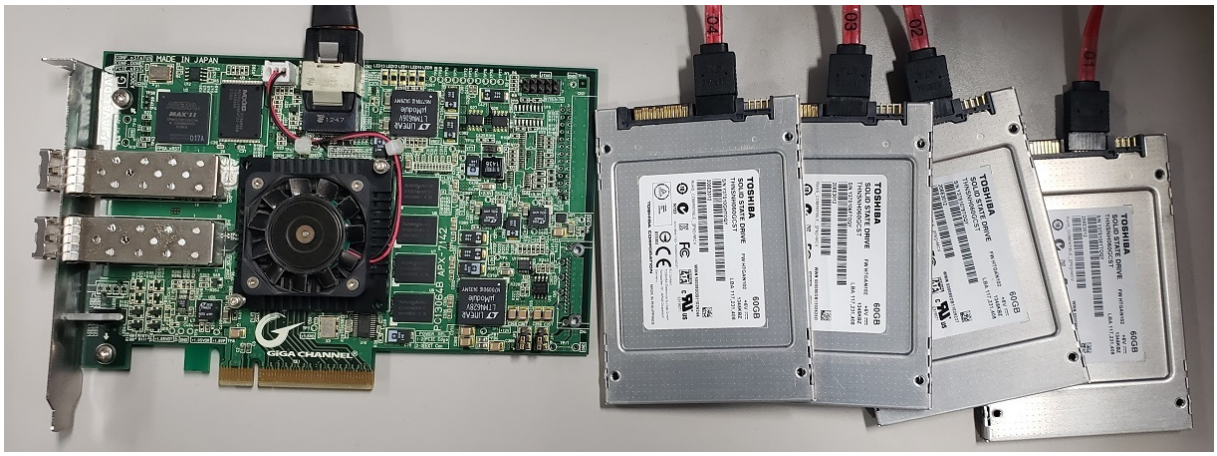

図5.1.1: FPGAボードAPX-7142改

表 5.1: FPGA ボードの仕様

Product	APX-7142 改
FPGA Device	Stratix V GX 5SGXMA3K1F40C2N
DRAM	DDR3 800 MHz, 2 GB
Flash Storage	SAS connecter extends 4 SATA ports TOSHIBA THNSNH060GCST 60GB × 4
Network	Proprietary GiGA CHANNEL Optical token ring network 14 Gbps × 2ch
PCIe I/F	Gen2 × 8 Lane
Internal Bus	Proprietary AVAL-bus 256 bits-width, 125MHz

5.2 FPGA 上に構成するハードウェア

FPGA 内のハードウェア構成を図 5.2.1 に示す．FPGA 内部の AVAL-BUSSW を介して各インタフェイスやモジュールが相互に接続されている．これらのうち，SATA I/F および PRA を実装し，それ以外のインタフェイスやモジュールはアバールデータ社の IP を使用している．PRA については，5.3 節で説明する．モジュール間のデータ転送の指示は，Host I/F から発行される DMA（Direct Memory Access）命令によって行われる．SSD に格納されたデータは，Host I/F から発行される DMA 命令によって SATA I/F を通じて読み出され，バスを介して DRAM I/F や NW I/F へと送出される．受信側ホストも DMA 命令を発行することによって，NW I/F から入力されるデータを SATA I/F を通じてフラッシュストレージに格納することができる．このデータ転送は，送受信ホスト上の主記憶を介さずに行うことができ，高速かつ低負荷なデータ通信が可能となっている．また，PRA は，SATA I/F とバスの間に組み込まれており，In-datapath Computing を実現している．PRA への命令を書き込むには，Host I/F からローカルレジスタを通じて PRA へ命令を送信する．

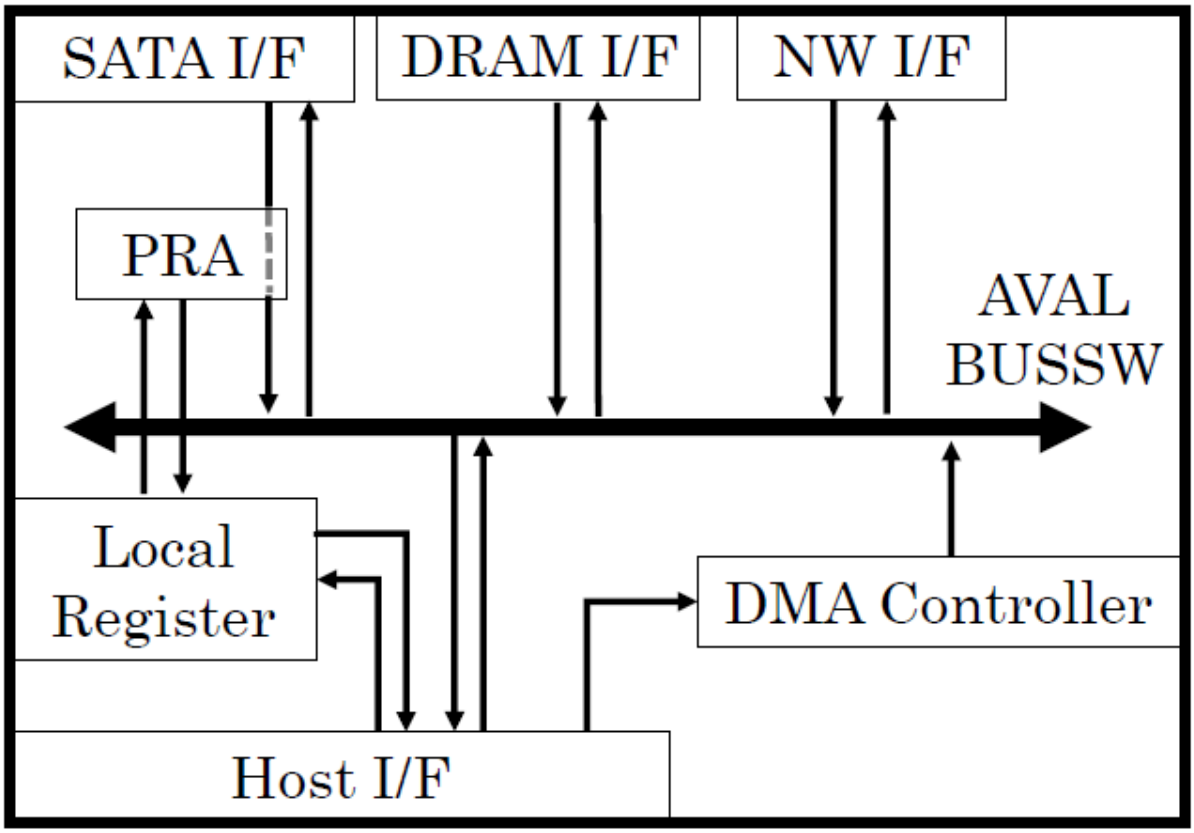

図 5.2.1: FPGA 内のハードウェア構成

5.3 PRA(Projection, Restriction, Aggregation)

4 節で述べたクエリ処理パイプラインを実現するハードウェア構成を図 5.3.1 に示す．パイプラインの 3 つのステージの処理機能をそれぞれ Projection (カラム抽出)，Restriction (タプル抽出)，Aggregation (集約演算) の 3 つのモジュールに実装する．各モジュールは書き換え可能な命令メモリを持ち，ホストからローカルレジスタを通じて受信した命令列を格納する．この命令列に従って，クエリを実行する．以上の 3 種のモジュール (Projection，Restriction，Aggregation) を統合したハードウェアモジュールを FPGA 上に実装する．この統合したハードウェアモジュールを各モジュールの頭文字から PRA と呼ぶ．各モジュールのハードウェア構成について以下に詳しく述べる．

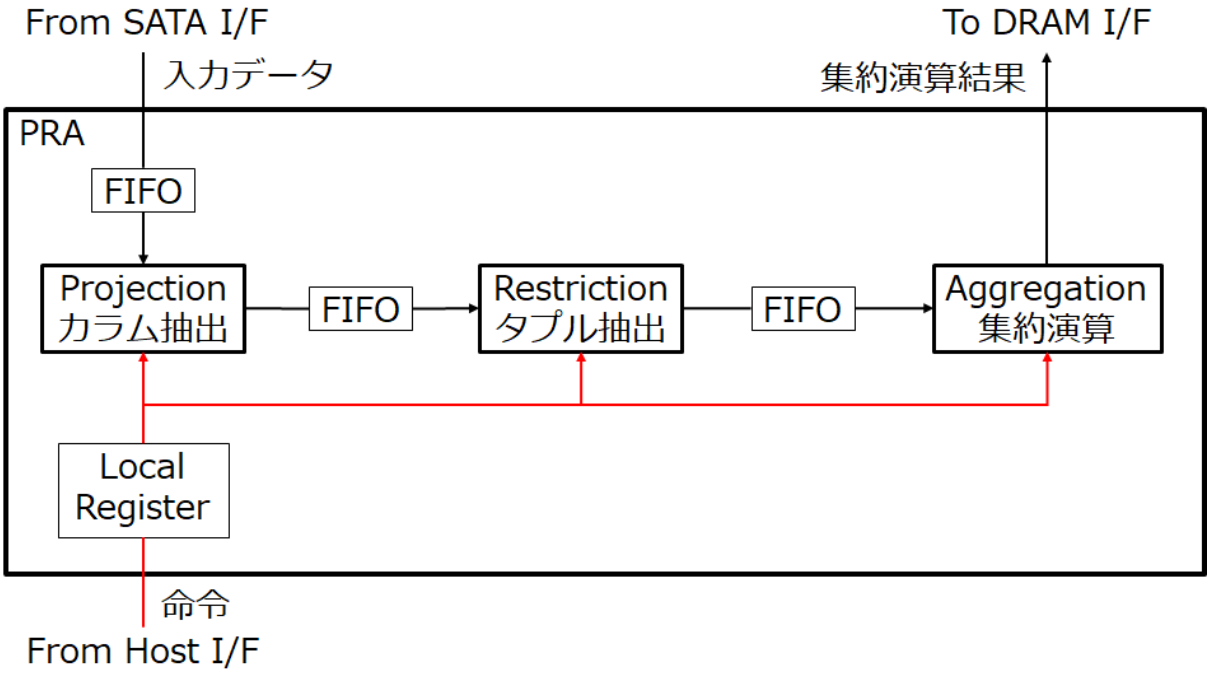

図 5.3.1: PRA モジュールのハードウェア構成図

5.4 Projection モジュールの実装

図 5.4.1 に Projection モジュールのハードウェア構成を示す．Projection モジュールへの入力データは，命令メモリから受信した命令どおりに Controller でデータのフィルタリングを実行し，カラムの抽出を行う．カラム抽出後のデータを FIFO に出力する．その後，Restriction モジュールへ出力する仕組みである．Projection モジュールへの命令は，1 チャンクごとに 0〜127 までの 128 ビットの命令列を図 5.4.2 に示す構成で設定する．ビット 127 は，1 タプルにおける最終チャンクか否かを判断し，ビット 126 はチャンクが必要か否かを判断する．命令列を 8 ビットごとに 16 分割し，そのうちの下位 5 ビットで必要なカラムかどうかと出力場所を決定する．下位 5 ビットのうち 5 ビット目でカラムが必要か否かを判断し，残りの 4 ビットでチャンク幅 16 バイトのうち，どのバイトに値を出力するかどうかを決定する．

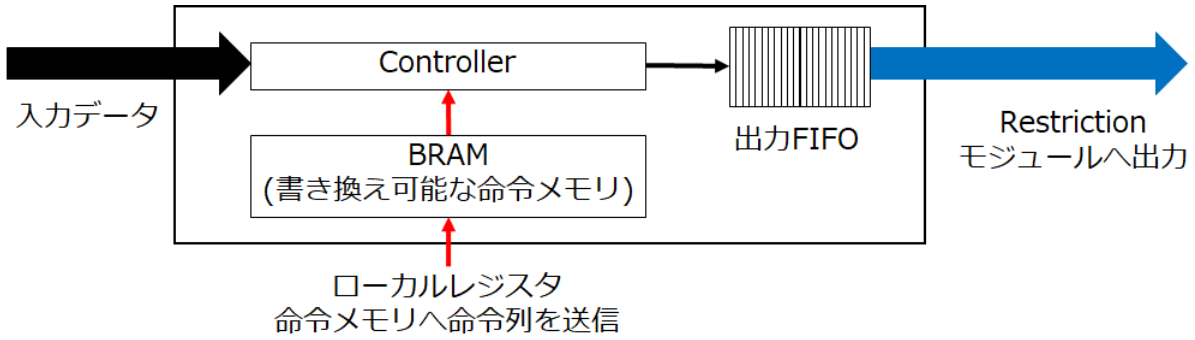

図 5.4.1: Projection モジュールのハードウェア構成図

bit127	bit126	6bits				8bits		
最終チャンク確認(1bit)	必要チャンク確認(1bit)	N/A (1bit)	必要カラム確認(1bit)	出力場所 (4bits)	・・・	N/A (3bits)	必要カラム確認(1bit)	出力場所 (4bits)

図 5.4.2: Projection モジュールへの命令列構成

5.5 Restriction モジュールの実装

図 5.5.1 に Restriction モジュールのハードウェア構成図を示す．Restriction モジュールは，複雑な SQL クエリを単純な命令に変換するために，MIPS アーキテクチャを使用する．Restriction に入力されたデータはまず，データタプルやその他の中間変数を格納するための 64 ビットレジスタの 2 つのバッファのどちらかに入力される．2 つのバッファを用意することで，一方のバッファで処理をしている間にもう一方のバッファで Projection モジュールからの入力を受け付け，スループットを向上させる．Restriction モジュールへ送信された命令は，命令メモリとして扱う BRAM に，命令の実行順が書かれた命令アドレスは PC ctrl と呼ばれる BRAM に書き込まれ，基本的には実行順通りに命令を発行する．ただし，分岐命令が読み出された際は分岐結果に従い，PC ctrl が順序とは異なる命令を発行する．RF Reg., EX Reg., WB Reg. はオペコードを読み取り，行う命令を決定する．また，Special Modules では，乗算のような 1 クロックサイクルで処理が終わらない命令を実行する．このモジュールで処理を行っている間でも他の命令を処理できる構造となっており，これによって計算効率を向上させることができる．Restriction モジュールへの命令は，ホストからローカルレジスタを通じて SSD からデータを読み出す前に設定する．設定する命令例を図 5.5.2 に示す．

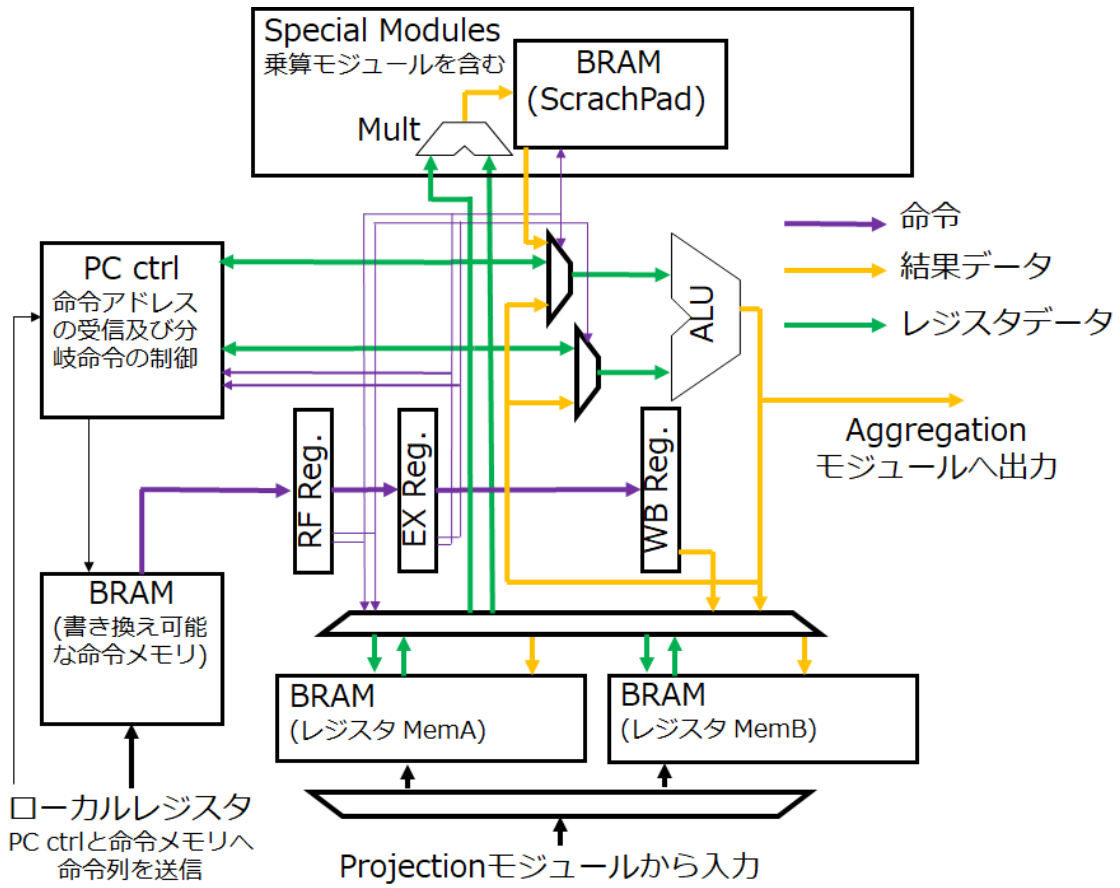

図 5.5.1: Restriction モジュールのハードウェア構成図

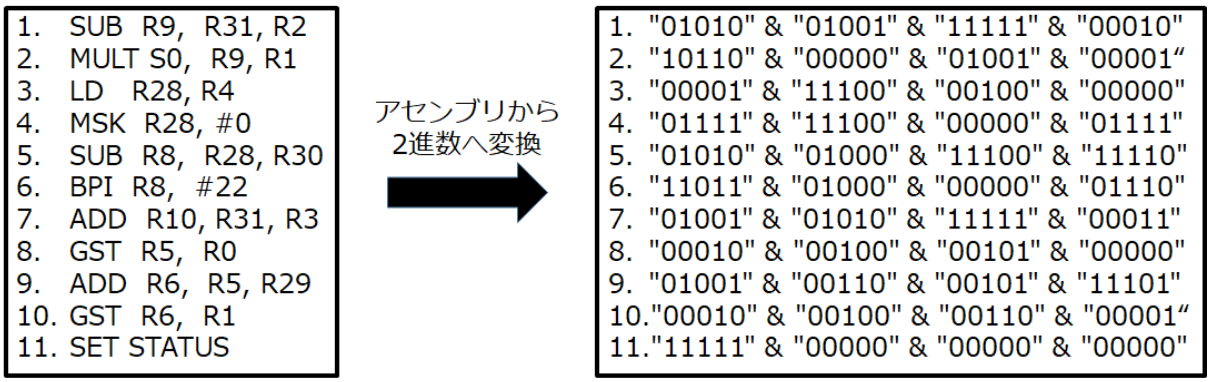

図 5.5.2: Restriction モジュールへの命令例

5.6 Aggregation モジュールの実装

図 5.6.1 に Aggregation モジュールのハードウェア構成図を示す．Aggregation パイプと呼ぶ集約演算 KEY によって使用するパイプを変更するようなパイプラインを実装する．これは，集約演算 KEY の種類が巨大かつ分布がランダムな場合，Aggregation モジュールでのキャッシュミスが増えるため，集約演算処理が律速になると考えられる．そこで，集約演算 KEY によって扱うパイプを変更することで，キャッシュミスを減らすことが可能となっている．本研究で扱う改良した TPC-H のクエリ 1 改およびクエリ 3 改は上記の条件に当てはまらないため，1 本のパイプでデータ処理を行うことが可能である．右図は，Aggregation パイプを 2 本で構成した図である．Aggregation モジュールは，集約演算結果を格納するために BRAM をキャッシュメモリとして使用している．動作方式は，キャッシュの更新を高速に行うためライトバック方式であり，データの確認にはダイレクトマップ方式を採用している．確認に用いる集約演算 KEY は Tag Array に格納されており，対応するデータは Data Array に格納されている．Restriction モジュールから出力されたデータと集約演算 KEY に対して，集約演算 KEY が一致するデータをキャッシュから読み出し，集約演算を行い結果をキャッシュに書き戻す．キャッシュミス時には，キャッシュに格納されたデータを DRAMC (DRAM Controller) を通じて FPGA 内部の DRAM へ書込み，集約演算 KEY に対応するアドレスのデータを DRAM から読み出す．最後に，必ずキャッシュミスするデータを書き込むことで，キャッシュからデータを DRAM へ追い出し，ホストから DRAM を確認することで，結果の確認を行う．

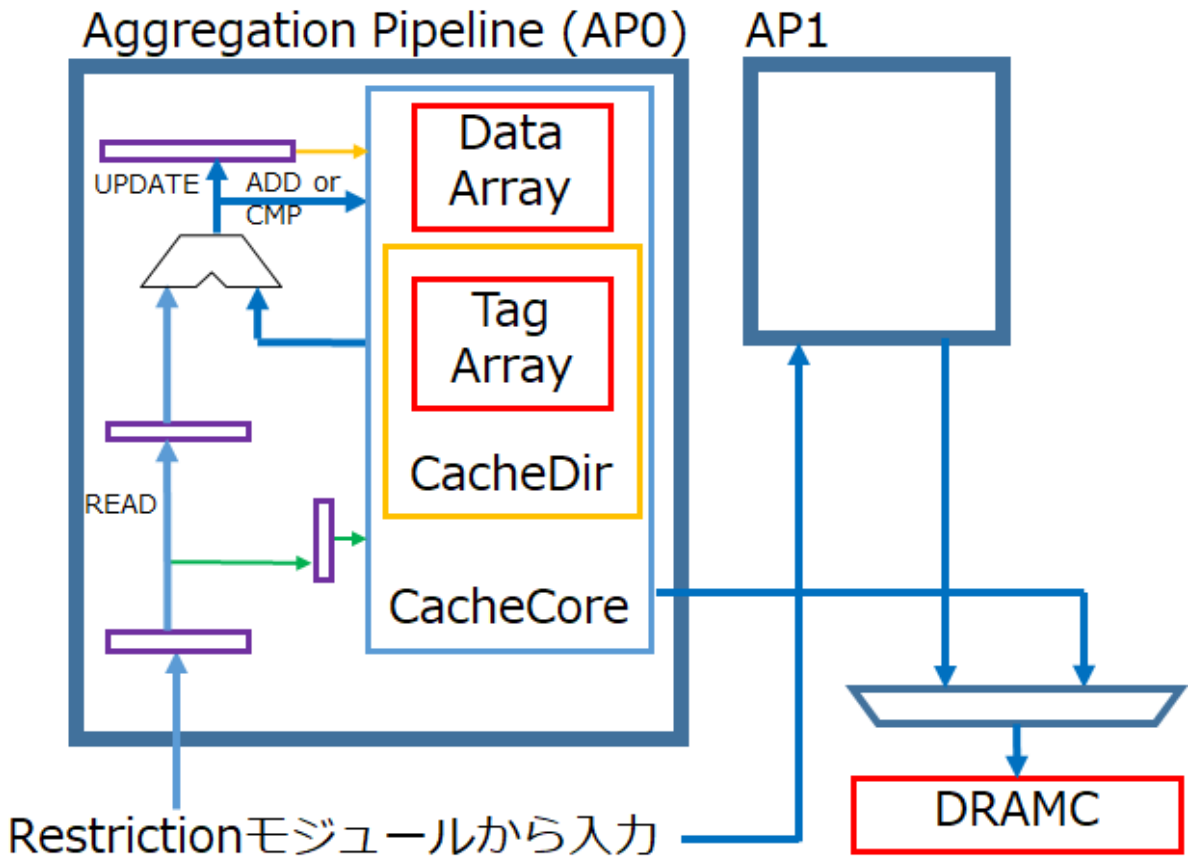

図 5.6.1: Aggregation モジュールのハードウェア構成図

第6章　評価

6.1　実験環境

6.1.1　計算機環境

表 6.1 に示す計算機環境を使用して 5.3 節で述べたハードウェアを実装した FPGA を最大 4 ノード用いて実験を行った．ホスト PC の外観は図 6.1.1 に示したものであり，ホスト PC1 台につき 1 つの FPGA ボードが搭載されている．また，比較対象として，表 6.2 に示す計算機環境を使用して，Impala によるソフトウェアのみでの実行を行った．

表 6.1: FPGA 実験環境

No. of Node	1 or 4	
CPU	Intel(R) Core(TM) i5-10400	2.90GHz (6C12T)
Memory	DDR4 2666 MHz	32GB
Network	Intel Ethernet Controller X540-AT2	10Gbps
	On board Ethernet	1Gbps
SSD	Transcend TS64GSSD370S	64GB
	Western Digital SSD WD Blue	1TB

表 6.2: ソフトウェア (Impala) 実験環境

No. of Node	1 or 4	
CPU	Intel Core i5-8400	2.80GHz (6C6T)
Memory	DDR4 2133 MHz	32GB
Network	ASUS 10G Network adapter XG-C100C	10Gbps
	On board Ethernet	1Gbps
SSD	Samsung SSD 860 QVO	500GB

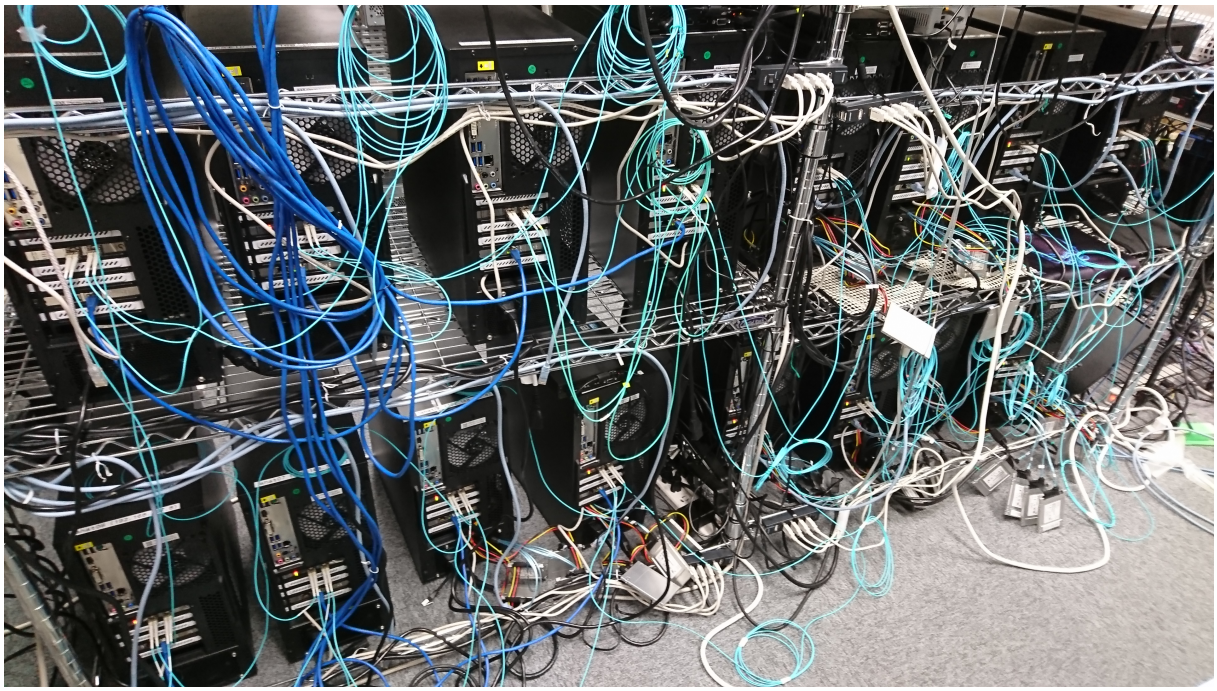

図 6.1.1: 実験に使用したホスト PC

6.1.2 評価クエリ

本実験の評価クエリとして，図 4.3.1 および図 6.1.2 に示すような OLAP のベンチマークの 1 つである TPC-H のクエリ 1 およびクエリ 3 の SQL 文を改良したものを用いた．クエリ 1 改では，前述したようにソフトウェア側で行う AVG の演算を省いた．また，図 6.1.3 に示すクエリ 3 の SQL 文では，customer および lineitem, orders と呼ばれる 3 つのテーブルの結合演算が必要となっている．しかし，本研究の実装ハードウェアにテーブルの結合演算機能が備わっていないため，事前に 3 つのテーブルで結合演算を行ったテーブル (lineitem_q3) を作成し，クエリ 3 改として実行した．TPC-H がベンチマークするデータ量を Scale Factor と呼び，1GB 単位で設定できる．Scale Factor を 2 のべき乗単位で増やし，実験を行った．また，評価クエリのタプル構成をそれぞれ図 6.1.4，図 6.1.5 に示す．黄色で塗られたカラムが必要なカラムを表している．必要なカラムがあるチャンクはなるべく少なくなるように，かつ集約演算 KEY として用いるデータは，同じチャンクに存在するように構成している．

```
SELECT
        l_orderkey,
        sum(l_extendedprice * (100 - l_discount)) / 10000 as revenue,
        o_orderdate, o_shippriority
FROM
        lineitem_q3
WHERE
        c_mktsegment = 'A'
        and o_orderdate < '1995-03-01'
        and l_shipdate > '1995-03-01'
GROUP BY
        l_orderkey, o_orderdate, o_shippriority
ORDER BY
        revenue desc, o_orderdate;
```

図 6.1.2: クエリ 3 改の SQL 文

```
SELECT
        l_orderkey,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        o_orderdate,
        o_shippriority
FROM
        customer,
        orders,
        lineitem
WHERE
        c_mktsegment = 'AUTOMOBILE'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < '1995-03-01'
        and l_shipdate > '1995-03-01'
GROUP BY
        l_orderkey,
        o_orderdate,
        o_shippriority
ORDER BY
        revenue desc, o_orderdate;
```

図 6.1.3: クエリ 3 の SQL 文

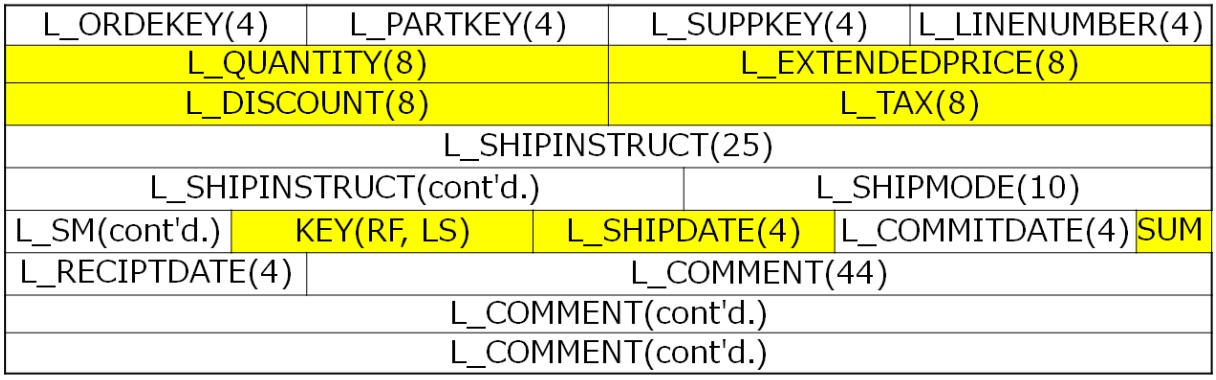

図 6.1.4: クエリ 1 のタプル構成

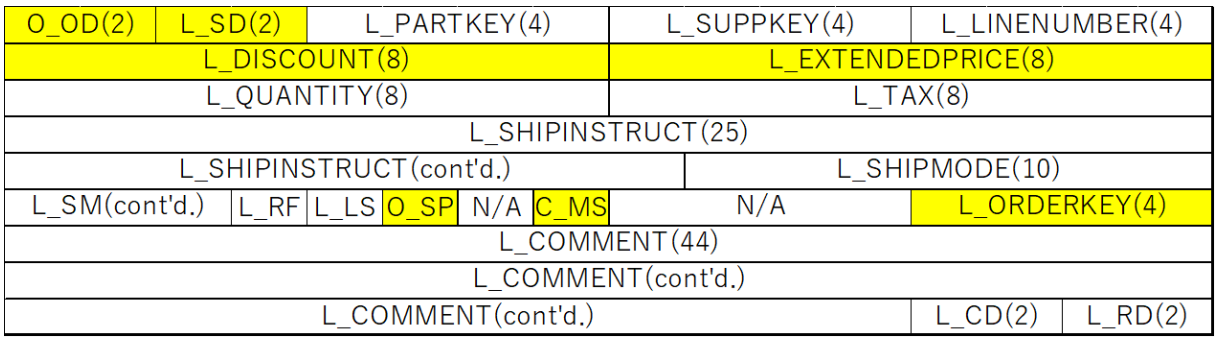

図 6.1.5: クエリ 3 改のタプル構成

6.2 予備評価

6.2.1 Restriction モジュールの多重化

5.5 説で説明した Restriction モジュールのスループットについて考察する．125MHz で稼働するデータ幅 16B の Restriction モジュールは，最大 2GB/s(=16[Bytes]/125[MHz]) のスループットを持つ．一方で，APX7142 改の場合は 4 並列の SSD で，約 1GB/s でデータが読み出される．このことから，Projection モジュールによって削減されたカラムのデータ速度が，Restriction モジュールへの入力待ちになる程度に演算量が多い場合，処理に時間がかかり律速となってしまう．そこで，Restriction モジュールを 2 つに増やし，図 6.2.1 に示すようなラウンドロビン方式での並列化を実現する実装を行う．Restriction モジュールを 1 つまたは 2 つ実装して実験を行い，比較した．実験結果を図 6.2.2 に示す．この実験は，クエリ 1 改を対象クエリとして行い，FPGA1 ノードで実験したものである．SSD 読み出しと同等のスループットでクエリが実行されていることが確認できた．

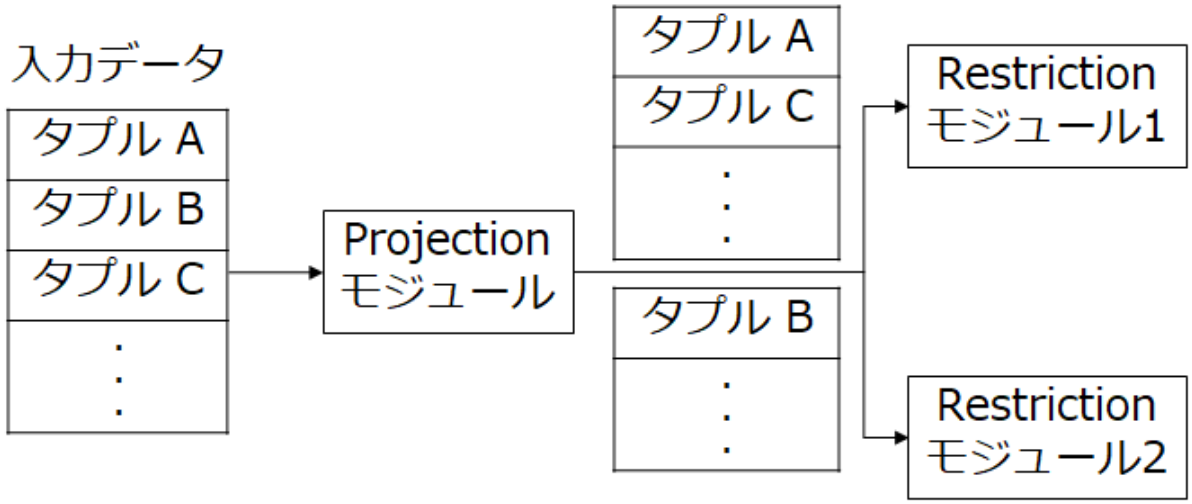

図 6.2.1: Restriction モジュール 2 つ構成のデータの流れ

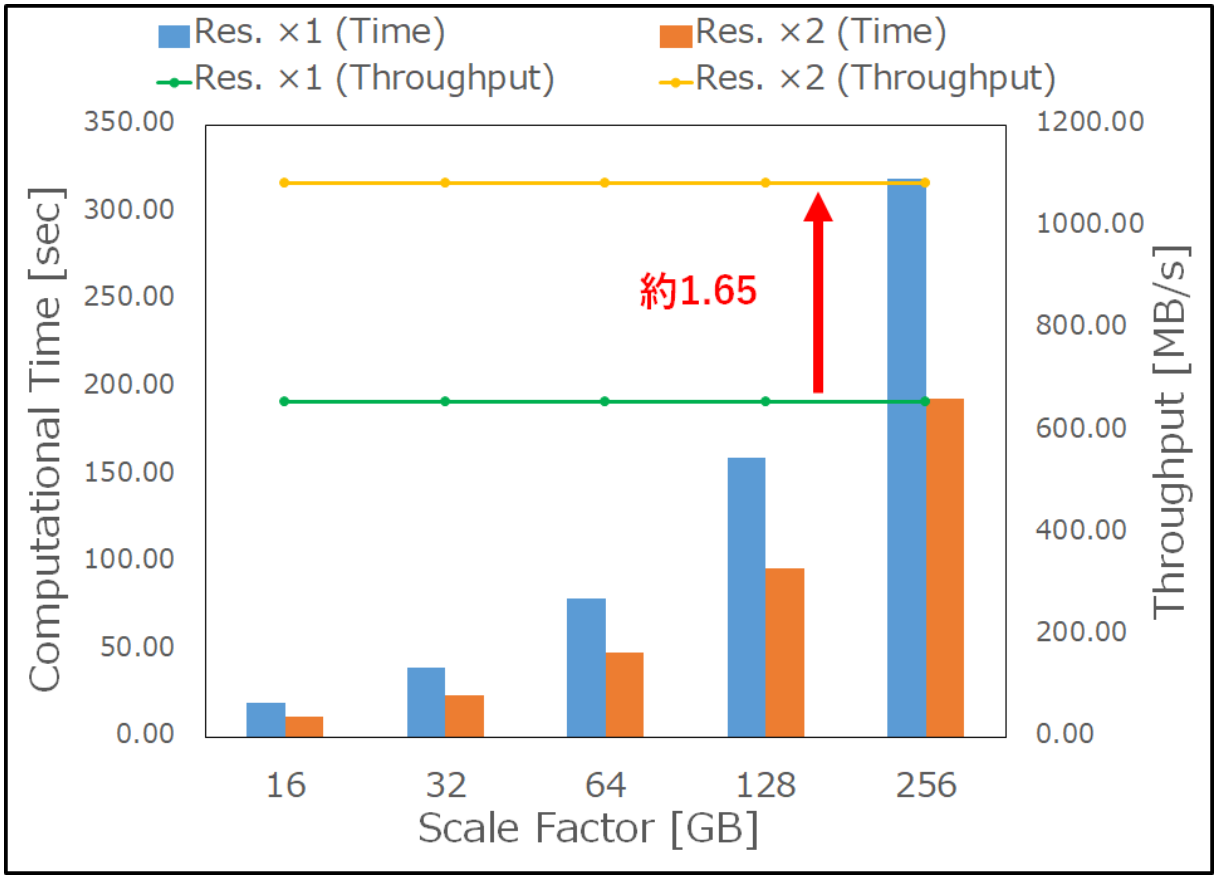

図 6.2.2: Restriction モジュール数と実行時間 (1 ノード)

6.2.2 ハードウェアリソースの使用率

APX-7142 改に搭載されている FPGA(Stratix V) に実装したときのハードウェアリソースの使用率を表 6.3 に示す．ここで示す全体の使用率は，AVAL DATA 社が提供している GiGA-CHANNEL 等の IP モジュール，および図 5.3.1 に示したクエリ処理用の専用ハードウェアモジュールとその接続部分を全て含んだものである．また，Restriction モジュールの使用率は，Restriction モジュールを 2 つ実装した場合の使用率であり，Aggregation モジュールの使用率は，Aggregation Pipeline を 2 本で構成した場合の使用率である．Intel 社の FPGA 設計・開発ソフトウェアである Quartus を使用して調べた“ Logic utilization ”, および“ Total block memory bits ”について示す．“ Logic utilization ” は Intel 社の FPGA 内で使用される論理要素である“ Adaptive Logic Module(ALM) ”の使用率であり，ALM は LUT や加算器，レジスタ等で構成され，これらを組み合わせることで処理回路を生成している．“ Total block memory bits ”は，FPGA 上に用意された Block RAM の使用率を表している．

Aggregation Pipeline 2 本実装時に全体の”Logic utilization” は 59.7%で，32 本実装した場合，” Logic utilization”は 97.2%となると考えられる．そのため，64 本実装しようとすると使用している FPGA のハードウェアリソースの上限を超えるため，本実装で可能な本数の上限は 32 本と考えられる．

表 6.3: ハードウェア使用率

	Logic utilizatoin (ALM) [%]	Total block memory bits [%]
全体	59.7	31.7
PRA	6.9	6.5
Projection	2.2	0.5
Restriction	2.2	4.7
Aggregation	2.5	1.3

6.3　ソフトウェア実行との比較

図 6.3.1 に，4 ノードでの Impala によるソフトウェア実行および Restriction モジュールを 2 つ実装した FPGA 実行をクエリ 1 改を対象として行った実験結果を示す．FPGA4 ノードでの実行は，4 分割したデータを各ノードに入力したデータで集約演算を行い，最後に結果をホストノードにまとめた (例えば，SF32GB の実験では，各ノード SF8GB のデータ量で集約演算を実行している)．また，図 6.3.2 にクエリ 3 改を対象とした 1 ノードでの Impala によるによるソフトウェア実行およびクエリ 3 改を対象とした Restriction モジュールを 1 つ実装した FPGA 実行による実験結果を示す．ソフトウェアでの実行速度と比較して，クエリ 1 改における実行速度は約 7.1 倍〜約 11.0 倍，クエリ 3 改における実行速度は約 10.3 倍〜約 10.7 倍となった．すなわち，Impala はインメモリで実行できない Scale Factor を超えると，メモリからデータの追い出しが発生しスループットが低下する．一方，FPGA は SSD からの読み出し速度で集約演算を実行し，Scale Factor に関わらず一定のスループットを維持するため，以上のような結果となった．

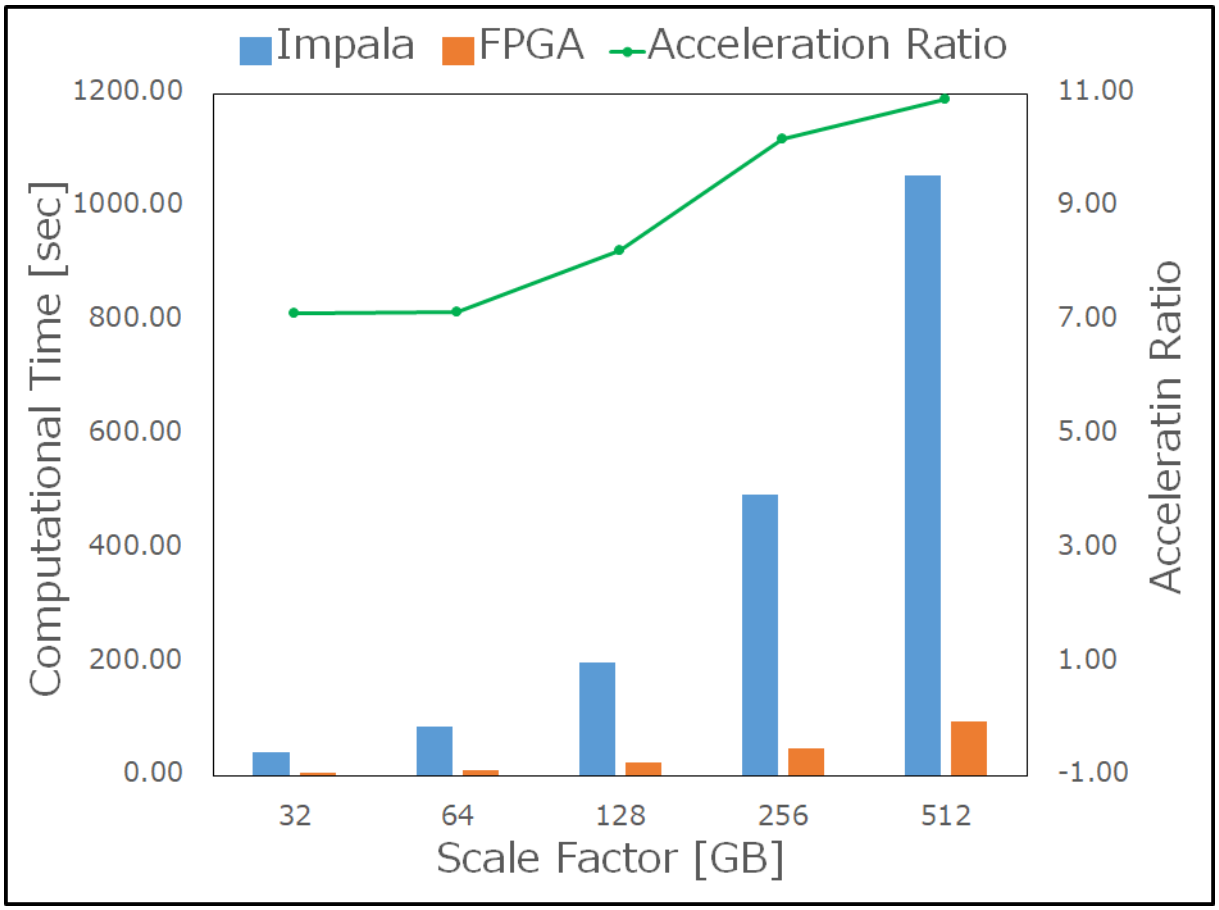

図 6.3.1: Impala と FPGA 実行の比較 (クエリ 1 改・4 ノード)

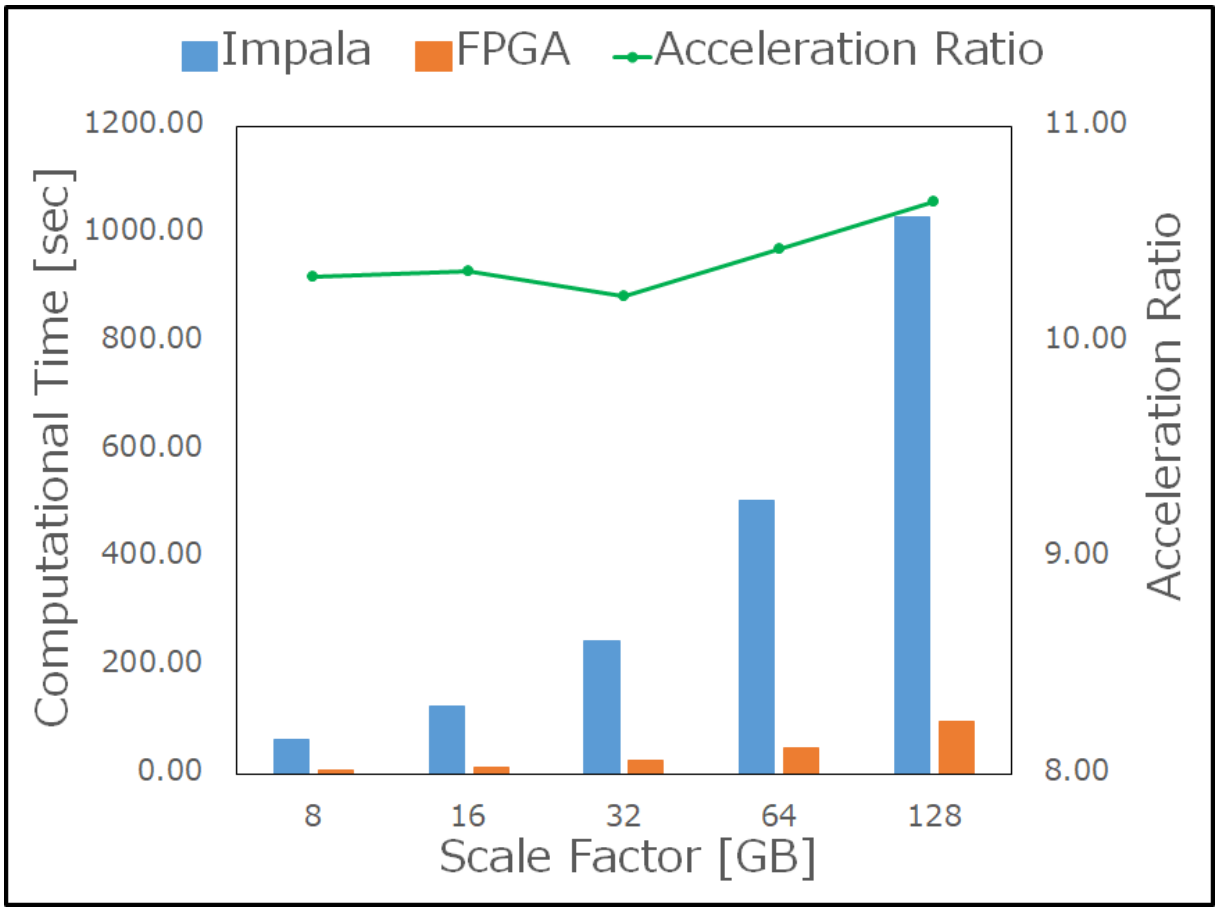

図 6.3.2: Impala と FPGA 実行の比較 (クエリ 3 改・1 ノード)

第7章　結論

本研究では，実行クエリを変更可能な集約演算ハードウェアの有効性を示した．実験の結果，Impala によるソフトウェア実行と比較した際に，約 7 倍以上の実行速度での実行が可能であり，ソフトウェア実行と異なり総物理メモリ量に関わらず，一定のスループットでの実行が可能であると確認した．

今後の課題としては，TPC-H のクエリ 1，クエリ 3 だけでなくより複雑なクエリでの実験やより多くのノード数での実験を行う．また，本ハードウェアには実装されていないテーブルの結合演算機能として PPJoin[12] と呼ぶ結合演算機能との接続を検討したい．

謝辞

本研究に際して，熱心なご指導を頂きました吉永努教授に感謝の意を表します．また，また, 研究内容に関して指摘をくださいました TIS 株式会社の吉見真聡氏に感謝致します. 最後に，研究生活を通じて様々な指摘，協力を下さいました吉永研究室の皆様に，厚く御礼申し上げます.

参考文献

[1] C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, Vol. 275, pp. 314 – 347, 2014.

[2] M. Yoshimi, R. Kudo, Y. Oge, Y. Terada, H. Irie, and T. Yoshinaga. Accelerating olap workload on interconnected fpgas with flash storage. In *2014 Second International Symposium on Computing and Networking*, pp. 440–446, 2014.

[3] 川原尚人, 吉見真聡, 策力木格, 吉永努. ネットワーク結合型マルチ fpga ボードを用いた集約演算クエリ処理. 信学技報, Vol. 116, No. 240, pp. 29–34, 2016.

[4] Transaction Processing Performance Council. TPC-H Revision 2.18.0, 2021. `http://www.tpc.org/tpch/`.

[5] Louis Woods, Zsolt István, and Gustavo Alonso. Ibex - an intelligent storage engine with support for advanced SQL off-loading. *PVLDB*, Vol. 7, No. 11, pp. 963–974, 2014.

[6] George S. Davidson, et al. Data-centric computing with the netezza architecture. *SANDIA REPORT*, pp. 1–24, Apr. 2006.

[7] Bo-Cheng C Lai, Chun-Yen Chen, Yi-Da Hsin, and Bo-Yen Lin. A two-directional bigdata sorting architecture on fpga. *IEEE Computer Architecture Letters*, pp. 72–75, 2020.

[8] Ying Li, Jinyu Zhan, Wei Jiang, Junting Wu, and Jianping Zhu. An fpga based network interface card with query filter for storage nodes of big data systems. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 556–561. IEEE, 2020.

[9] 飯塚健介, 天野英晴ほか. Alexnet のマルチ fpga システムへの分割検討と実装. 研究報告システム・アーキテクチャ (ARC), Vol. 2019, No. 14, pp. 1–5, 2019.

[10] Muhsen Owaida, David Sidler, Kaan Kara, and Gustavo Alonso. Centaur: A framework for hybrid cpu-fpga databases. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 211–218. IEEE, 2017.

[11] AVAL DATA. APX7142. https://www.avaldata.co.jp/products/network/item/apx-7142, 2014.

[12] Masato Yoshimi, Yasin Oge, and Tsutomu Yoshinaga. Pipelined parallel join and its fpga-based acceleration. *ACM Trans. Reconfigurable Technol. Syst.*, Vol. 10, No. 4, pp. 1–28, December 2017.

発表論文

[1] 尾作 洋彦，策力木格，吉見 真聡，吉永 努 “FPGA を用いたデータベースクエリ処理の高速化” 信学技報, vol. 120, no. 338, pp. 90-95, Jan. 2021.